Rapid Vocabulary Addition to Context-Dependent Decoder Graphs

Cyril Allauzen, Michael Riley

Google, Inc
allauzen@google.com, riley@google.com

Abstract

We describe how to efficiently add new vocabulary directly to an existing optimized ASR decoder graph. The augmented decoder graph is represented by two weighted finite-state transducers, a primary graph that represents the static portion and a secondary graph that represents the dynamic portion, together with a mapping that specifies that some states in the two graphs are to be merged. Determinism is obtained by excluding from the secondary graph any prefixes already present in the primary graph. Correct context-dependency is obtained by including in the primary graph all prefixes needed to properly merge with the secondary graph. We report experiments comparing this approach to an existing one that requires on-the-fly construction of the decoder graph.

Index Terms: speech recognition, decoding architecture, finite-state transducers

1. Introduction

We describe how to efficiently add new vocabulary to an existing optimized ASR decoder graph. We will assume this graph is represented as a weighted finite-state transducer (WFST), which has been shown to be a general and efficient representation in speech recognition [1, 2]. WFSTs can be used to represent a language model (an automaton over words), the phonetic lexicon $L^*$ (a context-independent-phone-to-word transducer), and the context-dependency specification $C$ (a context-dependent-phone to context-independent-phone transducer). Further, these can be combined and optimized with the finite-state operations of composition, determinization, and minimization as:

$$C \circ \min(\det(L^*)) \circ G$$

(1)

to produce a recognition transducer that is very efficient to search in ASR decoding.

While this recognition transducer is well-optimized for decoding as is, there are circumstances where we wish to modify the search graph just prior to recognition. One example would be to replace a non-terminal symbol such as $\$PERSON$ or $\$CITY$ in the grammar $G$ that represents a list of persons or cities relevant to the immediate recognition context. Another example, the one we address in this paper, is to add new pronunciations to the lexicon $L^*$ for previously out-of-vocabulary words (OOVs). This might be necessary, for instance, to cover some person or city that has been added for which there is no existing pronunciation. In these cases, there is not enough time to completely repeat the recognition transducer construction in Eq. 1 with these changes.

There are two general approaches to solve these dynamic decoder graph problems. The first is to build the recognition cascade on-the-fly during decoding [3, 4, 5, 2, 6]. This easily permits replacements of non-terminals in the grammar $G$ [7, 8, 9]. Since the lexicon is optimized, it is not as obvious how to add new vocabulary, but if the lexicon contains every monophone as an auxiliary monophone word then any new vocabulary can be spelled out in the grammar $G$ as a transduction that reads a sequence of monophone words on input, representing the new word’s pronunciation, and that outputs the new word label [8, 9]. A variant is to use a concatenation of existing word pronunciations to form the new word’s transduction in $G$ [7].

The second approach for dynamic decoder graph problems is to directly modify an optimized, statically constructed recognition transducer. This is non-trivial since the optimization of the lexicon and its composition with the context-dependency specification and grammar creates a complex automaton with considerable sharing of structure. The potential advantages of this approach are that any complexity and overhead associated with on-the-fly expansion during decoding is avoided and increased determinism may result with a suitable construction.

[10] took this second approach for the expansion of grammar non-terminals using auxiliary markers to ensure correct context-dependent substitutions. In this paper, we take a related approach for vocabulary additions to optimized lexicons. While this could be treated entirely as a special case of non-terminal replacement (e.g., $\$OOV$), we seek a result that preserves a fair degree of the determinism of the construction in Eq. 1 for efficient use. Carraco and Forcada [11] describe a method for incrementally adding a string to a minimal automaton. Our additions, however, are more complex.

In Section 2 we first consider adding new words to an optimized context-independent (CI) lexicon. In Section 3 we extend this to context-dependent (CD) lexicons. In Section 4 we then extend this to general decoder graphs. In Section 5 we compare the approach for new vocabulary described here to the on-the-fly method described in [9]. Finally, in Section 6 we discuss the results.

2. Context-Independent Lexicons

2.1. Example

Before we present the method more formally, we give a pictorial example of adding new vocabulary to the lexicon depicted in Figure 1. This base lexicon represents a transduction from phones $\{a, b, c, \#, w\}$ to words $\{w1, w2, w3, w4\}$. The auxiliary phone $\#_w$ marks word ends. The bold state is the initial state, the transitions are labeled with a phone on input and with a word or $\epsilon$ (a free move) on output. The final state is a double circle. For example, the pronunciation $abc\#_w$ corresponds to the word $w1$ since that can be read on the topmost successful path, i.e. one from the initial state to the final state.

The first step is to complete the lexicon so it accepts all possible word pronunciations as depicted in Figure 2. Now any pronunciation not found in the lexicon is transduced to the new word symbol $\nu$. We require each such new word to contain an
Figure 1: The base lexicon is a transducer from phonetic pronunciations to known word IDs.

auxiliary phone $\#_p$ that marks the end of prefixes that will be shared in the primary graph. In this example, the $\#_p$ phone demarks prefixes of depth 2. The depth is a parameter of the construction with tradeoffs that we describe later.

Figure 2: The completed lexicon accepts all possible word pronunciations. The primary graph is the sub-transducer depicted in black.

To obtain the primary graph we delete all paths after any $\#_p$; it is shown in black in Figure 2. Note this is similar to the base lexicon except it contains all prefixes of depth $d$ we might need when adding new vocabulary. By construction, it preserves the determinism of the base lexicon.\footnote{In general, it is only deterministic when considered as an automaton over the extended phone set, i.e., when input and output labels are considered as a single label and the auxiliary symbols are included.}

We also generate the affix transducer depicted in Figure 3 from the completed lexicon, which is used to determine which primary graph states are merged with states in the secondary graph, which will contain the new vocabulary. The affix transducer is constructed from the completed lexicon by deleting all non-OOV paths and relabeling each $\#_p$ and $\#_w$ transition to output its corresponding primary-graph source and destination state, respectively (using the notation $sn$ for the $n$-th state).

Figure 3: The affix transducer determines the secondary graph and how it is merged with the primary graph.

Figure 4 shows the new vocabulary we will add (namely, words $\{n1, n2, n3, n4\}$), which we also represent as a transducer.

Figure 4: The new vocabulary transducer maps from phonetic pronunciations to new word IDs.

Figure 5 shows how the affix transducer is used by composing its inverse with the new vocabulary transducer. We have used a three-tape composition that retains all three alphabets: transitions are labeled with a primary graph state label, a phone label (used in the composition match) and a word label. Each path contains exactly two (non-$\epsilon$) state labels on the first tape, which we will call the source and destination state labels, and one (non-$\epsilon$) word label on the third tape.

The secondary graph is formed from the sub-graph between the source and destination state labels (after moving the unique word label per path within those boundaries if necessary) and
is merged into the primary graph at the source and destination state-labeled states. This step is depicted in Figure 6.

2.2. Method

We begin by considering the simple case of a context-independent phonetic pronunciations, ignoring the word outputs for now. Assume we have a vocabulary of \( l \) words with phonetic pronunciations \( L = \{\varphi_1, \varphi_2, ..., \varphi_l\} \), \( \varphi_i \in \Phi^* \), where \( \Phi \) is the phonetic alphabet. By convention, we assume the last phone in the pronunciation is the auxiliary word-end symbol \( \#_w \). Let \( L \) be the minimal, deterministic automaton over alphabet \( \Phi \) that represents the finite set of strings \( L \).

Suppose we also have a vocabulary of \( n \) new words with phonetic pronunciations \( N = \{\varphi_{l+1}, \varphi_{l+2}, ..., \varphi_{l+n}\} \). By convention, we assume that the \( d \)-th phone in the pronunciation of the new words is the auxiliary prefix-end symbol \( \#_p \) and the last phone is \( \#_w \). Let \( N \) be the minimal, deterministic automaton representing \( N \).

We wish to combine these two lexicons to represent \( L \cup N \) in an incremental way and get a deterministic result. Consider the following identity:

\[
L \cup N' = (L \cup U_N) \cap (N \cup U_L)
\]  

for any sets \( U_L, U_N \subseteq \Phi^* \) such that \( L \subseteq U_L \), \( N \subseteq U_N \), and \( U_L \cap U_N = \emptyset \), which is easy to verify using distributivity. Take \( U_L = \Sigma^* \{\#_w\} \) and \( U_N = \Sigma^* \{\#_p\} \Sigma^* \{\#_w\} \) where \( \Sigma = \Phi - \{\#_p, \#_w\} \). Let \( \hat{L} \) be the minimal, deterministic automaton representing \( L \cup U_N \), which we call the completed base lexicon (cf. Figure 2). Similarly, let \( \hat{N} \) be the minimal, deterministic automaton representing \( N \cup U_L \).

We can categorize states in \( \hat{L} \) as follows: (1) those that are on successful paths labeled with a string in \( L \) are called base states, (2) those on successful paths labeled with a string in \( U_N \) and preceding the \( \#_p \) or following the \( \#_w \) are called affix states\(^3\), (3) otherwise they are called stem states. The sub-automaton of base and affix states forms the primary graph (cf. Figure 2) while the sub-automaton of affix and stem states forms the affix automaton \( \hat{A} \).

We seek \( \text{det}(L \cup N) \), the deterministic combined lexicon, which is equivalent to \( \hat{L} \cap \hat{N} \) by Eq. 2 and is deterministic by construction.\(^4\) Each state in \( \hat{L} \cap \hat{N} \) corresponds to a pair of states one from each input to the intersection [1]. These states can be decomposed into those that pair with the primary graph states and those that otherwise form the secondary graph (cf. Figure 6). The states of the secondary graph correspond to the states in \( N \cap \hat{A} \) that pair with stem states of \( \hat{A} \). Promote \( \hat{A} \) to a transducer that outputs the corresponding source state ID in \( L \) when entering the stem states (at the \( \#_p \)) and the destination state ID when exiting the stem states (at the \( \#_w \)) to create the affix transducer \( \hat{A} \) (cf. Figure 3). From \( N \circ \hat{A} \), the secondary graph can be identified along with how to merge it with the primary graph by the location of the state ID output labels (cf. Figure 5).\(^5\) In the context-independent case, the prefix depth \( d \) trades off the possible sharing of new words with the base lexicon with the size of the primary graph.

We used Eq. 2 to perform deterministic union with a form of completion and intersection. A more standard presentation would use DeMorgan’s Law [11]. However in the more general case, we must close the lexicon (to accept zero or more pronunciations) after adding the new pronunciations. Fortunately, a generalization of Eq. 2, \( (L \cup N)^* = (L \cup U_N)^* \cap (N \cup U_L)^* \), also holds since each word is terminated by a word end symbol \( \#_w \) in each of these sets. This identity can easily be verified by induction on the number of words and leads to a very similar lexicon construction.

The generalization to lexicon transducers that output word labels is straight-forward except the need to do the composition with the affix transducer in a way that all three alphabets are preserved and word labels may need to be moved back to the secondary graph as discussed in the example.

\(^3\)Some states, such as the initial state, may be both base and affix states.

\(^4\)The finite-state intersection of deterministic automata is deterministic.

\(^5\)As shown in the example, the \#_p marker can be left out of \( N \) and \( \hat{A} \) at this point for simplicity.
3. Context-Dependent Lexicons

We now consider the case of a context-dependent lexicon represented as an automaton of the context-dependent phonetic pronunciations, ignoring the word outputs. Assume we are given a CD transducer \( C \) that maps from context-dependent phones to context-independent phones \([1,5]\). Then \( L^*_n = \pi_n[C \circ L^*] \) is the context-dependent base lexicon formed by composing the CD transducer with the context-independent (closed) base lexicon \( L^* \) and projecting the result onto the input (context-dependent) tape \([1]\). We similarly define \( N^*_n = \pi_n[C \circ N^*] \) for the new vocabulary. We seek:

\[
\pi_n[C \circ \text{det}(L^* \cup N^*)] = \pi_n[C \circ (L^* \cap N^*)] \\
= \pi_n[C \circ L^*] \cap \pi_n[C \circ N^*] \\
\equiv L^*_n \cap N^*_n
\]

where we have used the results of Section 2.2. The definition there of base, suffix and stem states straightforward generalizes to \( L^*_n \) as does the construction of a (now context-dependent) primary graph and an affix transducer \( \hat{A} \). Like before, \( \hat{A} \) is used to construct the context-dependent secondary graph and specify how to merge it with the primary graph. In the context-dependent case, if the prefix depth \( d \) is greater than or equal to the longest history in the CD transducer (typically 2 for triphones models), then each state in the secondary graph will merge with at most one state in the primary graph (as in the context-independent case).

The treatment of lexicon transducers with word outputs is again very similar to the context-independent case.

4. Decoder Graphs

Assume an \( n \)-gram model \( G \) over an alphabet \( \Gamma \) that includes a unigram OOV word \( \nu \). The context-independent decoder automaton, ignoring words outputs, is formed by composing the context-independent closed lexicon with the grammar \( G \) and projecting onto its input tape. We seek:

\[
\pi_n[\tau[(L \cup N)^*] \circ G] = \pi_n[\tau[\hat{L}^* \cap \hat{N}^*] \circ G] = \pi_n[N^* \circ \tau[\hat{L}^*] \circ G] = \pi_n[N^* \cap \pi_n[\tau[\hat{L}^*] \circ G] = \pi_n[N^* \cap D]
\]

where \( \tau \) promotes a lexicon to a transducer by outputting the appropriate symbol from \( \Gamma \) for each word pronunciation. \( D \), the completed decoder graph, can be decomposed into primary, suffix and stem states and a primary graph and affix transducer formed like before, with the latter used to form the secondary graph. The generalizations to the transducer case with word outputs and to context-dependent case are straight-forward.

5. Experiments

To evaluate the method described here, we performed experiments with the voice search system described in \([9]\) modified as follows. First, the base lexicon is limited to the 10,000 most frequent words (thereby creating significant OOVs) and has 40 million \( n \)-grams. Second, we add from 100 up to 5,000 words to the base lexicon using two methods. The novel method described here, labeled \( CLG+OOV \), adds directly to the statically-constructed context-dependent decoder graph. The previously proposed method, labeled as \( G+OOV \), spells out at the unigram state in the grammar \( G \) each new word as a transduction from its phonetic pronunciation to its word label, adds auxiliary monophone words to the lexicon, and then dynamically expands the context-dependent decoder graph during recognition \([8,9]\).

Third, for both methods we generate lattice output and rescore with an LM that has a 15,000 word vocabulary and 50 million n-grams.

For a test set, we used 20,140 utterances (85,409 words) randomly drawn from anonymized Google mobile voice search traffic in the United States. Table 7 compares the two methods for adding new vocabulary.

<table>
<thead>
<tr>
<th>New Words Added</th>
<th>CLG+OOV WER (%)</th>
<th>CLG+OOV RTF</th>
<th>G+OOV WER (%)</th>
<th>G+OOV RTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>29.9</td>
<td>0.50</td>
<td>29.9</td>
<td>0.50</td>
</tr>
<tr>
<td>100</td>
<td>29.7</td>
<td>0.53</td>
<td>29.8</td>
<td>0.51</td>
</tr>
<tr>
<td>250</td>
<td>29.3</td>
<td>0.53</td>
<td>29.8</td>
<td>0.51</td>
</tr>
<tr>
<td>500</td>
<td>29.7</td>
<td>0.52</td>
<td>29.7</td>
<td>0.52</td>
</tr>
<tr>
<td>1000</td>
<td>28.7</td>
<td>0.54</td>
<td>29.4</td>
<td>0.50</td>
</tr>
<tr>
<td>5000</td>
<td>27.0</td>
<td>0.54</td>
<td>27.4</td>
<td>0.51</td>
</tr>
</tbody>
</table>

Figure 7: Results: Word-error rate (WER) and real-time factor (RTF) after adding new words to a 10K lexicon using two different methods - into the CD decoder graph directly (\( CLG+OOV \)) vs. as a phonetic transduction in the grammar \( G+OOV \).

For the fixed decoder beam used in all tests, the real-time factor did not change appreciably for any test condition. However, the \( CLG+OOV \) system gave significantly better accuracy compared to the \( G+OOV \) for each of the new vocabulary tests.

6. Discussion

The near constant real-time factor in the experiments likely reflects that the number of new unigrams is small relative to the overall number of \( n \)-grams. The accuracy improvement in the \( CLG+OOV \) is probably due to its better determinism compared to \( G+OOV \). In the \( G+OOV \) construction, new words share no prefixes with existing words in the decoder graph \([9]\) while in the \( CLG+OOV \) construction there is sharing up to the prefix depth \( d \). This means that the (lookahead) composition of the lexicon with the decoder graph, which pushes the weights as forward as possible \([2]\), will have a new word prefix weighted with a combination of its OOV weight and any matching higher probability, base lexicon words in the \( CLG+OOV \) system. Such paths are more likely to survive pruning during decoding compared to the \( G+OOV \) system and give rise to better accuracy for a given beam.

The primary graph can be constructed well before its use in recognition and the secondary graph construction is not computationally expensive (60 ms to build from an OOV lexicon of 5,000 words). It is the computational cost of their combination into a single decoder graph that is of principal interest here.

7. Acknowledgments

We thank Johan Schalkwyk for suggesting this approach and for valuable discussions.
8. References


